Diagonal Markowitz Scheme With Local Symmetrization*

Patrick R. Amestoy† Xiaoye S. Li‡ Esmond G. Ng§

August 1, 2005

Abstract

We describe a fill-reducing ordering algorithm for sparse, nonsymmetric **LU** factorizations, where the pivots are restricted to the diagonal and are selected greedily. The ordering algorithm only uses the structural information. Most of the existing methods are based on some type of symmetrization of the original matrix. Our algorithm exploits the nonsymmetric structure of the given matrix as much as possible. We show that our algorithm can be implemented in space bounded by the number of nonzero entries in the original matrix, and has the same time complexity as the analogous algorithms for symmetric matrices. We provide numerical experiments to demonstrate the ordering quality and the runtime of the new ordering algorithm.

Categories and Subject Descriptors: G.1.3 [Numerical Analysis]: Numerical Linear Algebra – *linear systems (direct methods), sparse and very large systems* G.4 [Mathematics of Computing]: Mathematical Software – *algorithm analysis, efficiency*

General terms: Algorithms, Experimentation, Performance

Keywords: sparse nonsymmetric matrices, linear equations, ordering methods

1 Introduction

We consider the direct solution of sparse linear equations $\mathbf{Ax} = \mathbf{b}$ using Gaussian elimination, where $\mathbf{A}$ is an $n \times n$ nonsymmetric sparse matrix. A major difficulty with nonsymmetric matrices is that they are rarely diagonally dominant, which means that during numerical factorization one must compromise fill-in reduction with numerical stability. Many nonsymmetric solvers deal with this situation using the *three-phase approach*, which includes an *analysis* phase, a *numerical factorization* phase, and a *triangular solution* phase [2, 6, 16]. Iterative refinements may be included in the triangular solution. The analysis phase includes a (numerical) preprocessing of the matrix and a symbolic phase that builds the computational graph for the numerical factorization phase. An advantage of the three-phase approach lies in its ability to anticipate the

*This work was supported in part by the Director, Office of Advanced Scientific Computing Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098, and in part by the NSF-INRIA Grant NSF-INT-0003274.

†ENSEEIHT-IRIT, 2 rue Camichel, 31071 Toulouse, France. email: amestoy@enseeiht.fr. Part of the work of this author was performed while he was on a sabbatical visit to NERSC.

‡Lawrence Berkeley National Laboratory, One Cyclotron Road, Mail Stop 50F-1650, Berkeley, CA 93720. USA. email: xsli@lbl.gov

§Lawrence Berkeley National Laboratory, One Cyclotron Road, Mail Stop 50F-1650, Berkeley, CA 94720. USA. email: EGNg@lbl.gov.

choice of the next pivot, which decouples the analysis from factorization and makes parallelization of numerical factorization easier. Therefore, it is a very important class of methods on high performance computers. In this context, it has been observed in [3] that it is critical to put numerically large entries on the diagonal during the preprocessing phase. One may then want to preserve this diagonal during sparsity reordering. That is, only a symmetric permutation is allowed afterward. One common practice to obtain such an ordering is to apply a symmetric ordering algorithm, either minimum-degree or nested-dissection variant, to the symmetrized pattern of $A + A^T$. Such reordering algorithms do not exploit the fact that during factorization the solvers can exploit the asymmetry of the permuted matrix.

In this article, we propose a new symmetric ordering algorithm. It is based on greedy heuristics that preserves the large diagonal entries and at the same time takes into account the asymmetry of the matrix. In the symmetric case, minimum-degree algorithm is a very effective greedy heuristic for fill-in reduction. By using the quotient graph elimination model [12, 13] and the approximate degree updates [1], the minimum-degree algorithm can be implemented very efficiently both in time and space. The nonsymmetric variant of minimum-degree was actually discovered earlier and was named after Markowitz [18], in which the "degree" of a vertex is the product of the row count and column count (known as the Markowitz count). But the original Markowitz algorithm is asymptotically slower than the minimum-degree algorithm, mainly due to the lack of a concise quotient graph model. A theoretical advancement was made by Pagallo and Maulino [20], who extended the quotient graph idea for symmetric matrices to the nonsymmetric case by introducing the *bipartite quotient graph* and showed that the bipartite quotient graph model can be implemented in space bounded by the size of $\mathbf{A}$. But timewise, using only the quotient graph model does not lead to an ordering algorithm that is as fast as the minimum-degree algorithm. This is because the lengths of the reachable paths to be searched when updating the Markowitz counts are not bounded. One main innovation of our work is the introduction of a *local symmetrization* mechanism that bounds the lengths of the reachable paths as in the symmetric case while capturing most of the asymmetry in the matrix. A secondary innovation is to extend and adapt the metrics to select pivots based on approximate degree [1] to metrics based on approximate Markowitz count and deficiency [22, 19]. Indeed, in our context all metrics have to anticipate the effect that local symmetrization would have on the pivot to be selected. Our algorithm has the same asymptotic complexity as the minimum-degree algorithm, both in space and in time.

The remainder of the paper is organized as follows. In Section 2, we first briefly introduce the bipartite quotient graph notation and properties. We then present the local symmetrization technique and describe our new ordering algorithm, in particular, how to update the quotient graph and how to compute metrics to select pivots within this framework. Section 3 describes the numerical experiments we have performed, and analyzes the effect of the new ordering algorithm on the multifrontal code `MA41_UNS` [2, 6]. Section 4 provides a summary of this research.

2 Diagonal Markowitz with local symmetrization

This section presents the algorithmic ingredients of our new Markowitz ordering framework. We show that the Markowitz algorithm can be implemented as efficiently as the approximate minimum-degree algorithm by using bipartite quotient graphs, the local symmetrization scheme, and the metrics based on approximate row and column degrees.

2.1 The Markowitz criterion

The Markowitz ordering algorithm [18] has been used successfully in general-purpose solvers [11]. This local greedy strategy can be described succinctly as follows. After k steps of Gaussian elimination, let r_i^k (resp. c_j^k) denote the number of nonzero entries in row i (resp. column j) of the remaining $(n-k) \times (n-k)$ submatrix. The (structural) Markowitz criterion is to select, as the next pivot, a nonzero entry a_{ij}^k from the remaining submatrix that has the minimum Markowitz count $(r_i^k - 1) \times (c_j^k - 1)$. This attempts to minimize an upper bound on the amount of fill-in generated at step $k+1$. Note that, in our context, we want to restrict the pivot selection to the diagonal of the remaining submatrix. This restriction of the Markowitz scheme will be referred to as the *diagonal Markowitz* scheme.

The simple rule above for choosing the next pivot does not immediately render an efficient implementation, because it requires updating the sparsity pattern of the remaining submatrix at each step, which may generate fill-in. From the development of the minimum-degree algorithm, which can be considered as a symmetric variant of Markowitz algorithm, we learned that by using the quotient graph elimination model [13], the algorithm can be implemented in space bounded by the size of the original matrix rather than that of the filled matrix. This is the so-called *in-place* property, and is very much desirable in an efficient ordering algorithm. Pagallo and Maulino [20] extended the quotient graph model to the *bipartite quotient graph* to model the nonsymmetric elimination, and showed that this model indeed has the in-place property. Now we briefly review this concept and illustrate how we can use and modify this model to design our ordering algorithm.

2.2 Bipartite quotient graphs

Let $\mathbf{A}$ be a nonsymmetric n-by-n matrix. The nonzero pattern of $\mathbf{A}$ can be represented by a bipartite graph $G = (V_r, V_c, E)$, where V_r and V_c are the sets of row and column vertices, respectively. For a row vertex $r_i \in V_r$ and a column vertex $c_j \in V_c$, an edge $(r_i, c_j) \in E$ exists if and only if $a_{ij} \neq 0$. Let $G^0 = (V_r^0, V_c^0, E^0)$ be the same as G. We use a bipartite graph $G^k = (V_r^k, V_c^k, E^k)$ to represent the nonzero pattern of the remaining submatrix after k steps of Gaussian elimination. Assuming pivots are chosen on the main diagonal, at step k, the transformation from G^{k-1} to G^k is based on the following elimination rule. Suppose the kth pivot node (r_p, c_p) is selected for elimination. The vertex sets become $V_r^k = V_r^{k-1} \backslash \{r_p\}$ and $V_c^k = V_c^{k-1} \backslash \{c_p\}$. The edge set E^k is derived from E^{k-1} by deleting the edges incident with c_p and r_p, and adding edges (r_i, c_j) for all r_i and c_j who are adjacent to c_p and r_p. This creates a fully connected bipartite subgraph (a *clique* in the symmetric analogue). We may refer to this as a *bipartite clique*, or *bi-clique* in short.

We now briefly review the (symmetric) quotient graph elimination model. The main idea is to use a compact representation to implicitly store the subgraph induced the vertices that have been eliminated. Suppose G is a undirected graph corresponding to a sparse symmetric matrix. Let S denote the subset of vertices in G that have been eliminated. Consider the subgraph $G(S)$ induced by S in G. In the quotient graph model, each *connected component* in $G(S)$ will be represented by a single "supervertex". As a result, any path in G from a vertex $i \notin S$ to a vertex $j \notin S$ through S corresponds to a path through at most one supervertex in S. The set of vertices adjacent to i in the remaining filled subgraph is given precisely by the reachable set of i through S. See [13] for details.

We now describe the nonsymmetric elimination process using the bipartite quotient graph model. We will use calligraphic letters to denote the sets associated with the bipartite quotient graph. Let $\mathcal{G}^k$ denote the bipartite quotient graph which represents the structure of the reduced submatrix after k steps of Gaussian elimination, and define $\mathcal{G}^0 = G^0$. When there is no ambiguity, we will omit superscript k. Both row and column vertices are partitioned into two sets: the set of uneliminated vertices referred to as *variables* and the set of eliminated vertices referred to as *elements*. That is, $\mathcal{G} = (\mathcal{V}_r \cup \bar{\mathcal{V}}_r, \mathcal{V}_c \cup \bar{\mathcal{V}}_c, \mathcal{E} \cup \bar{\mathcal{E}})$. Members of $\mathcal{V}_r$ ($\mathcal{V}_c$) will be referred to as *row (column) variables* (to distinguish them from the row vertices in V_r (V_c)), while members of $\bar{\mathcal{V}}_r$ ($\bar{\mathcal{V}}_c$) will be referred to as *row (column) elements*. The edge set $\mathcal{E}$ contains the edges between row (column) and column (row) variables. The edge set $\bar{\mathcal{E}}$ contains the edges between row (column) variables and column (row) elements, as well as the edges between row elements and column elements. An eliminated pivot $e = (r_e, c_e)$ has two vertices $r_e \in \bar{\mathcal{V}}_r$ and $r_c \in \bar{\mathcal{V}}_c$ referred to as a *coupled element*. Similarly an uneliminated pivot entry (a diagonal entry in the reduced matrix) $i = (r_i, c_i)$ will be referred to as a *coupled variable*. A nonzero entry (r_i, c_j) exists in the factors if and only if there exists a path of the form $r_i \to c_{e_1} \to r_{e_1} \ldots \to c_{e_l} \to r_{e_l} \to c_j$, where $e_i = (r_{e_i}, c_{e_i}), 1 \leq i \leq l$ are the coupled elements associated with the pivots already eliminated [21]. Therefore, following such paths, we can determine the nonzero entries of any row i or column j in the reduced submatrix.

Let $\mathcal{A}_{i*}$ be the set of column variables adjacent to row variable r_i in $\mathcal{G}$ which have never been modified after k steps of elimination. $\mathcal{A}_{*j}$ is defined similarly for column variable c_j. For each row variable r_i and column variable c_j, define the element adjacency lists:

$$\begin{aligned} \mathcal{R}_i &\equiv \{e = (r_e, c_e) : (r_i, c_e) \in \bar{\mathcal{E}}\} \subseteq \bar{\mathcal{V}}_c, \text{ the set of coupled elements adjacent to } r_i\,, \\ \mathcal{C}_j &\equiv \{e = (r_e, c_e) : (r_e, c_j) \in \bar{\mathcal{E}}\} \subseteq \bar{\mathcal{V}}_r, \text{ the set of coupled elements adjacent to } c_j\,. \end{aligned}$$

The adjacency lists of variables in the current bipartite quotient graph is then defined as:

$$\mathcal{U}_i \equiv Adj_{\mathcal{G}}^{row}(r_i) = \mathcal{A}_{i*} \cup \mathcal{R}_i\,, \tag{1}$$

$$\mathcal{L}_j \equiv Adj_{\mathcal{G}}^{col}(c_j) = \mathcal{A}_{*j} \cup \mathcal{C}_j\,. \tag{2}$$

For each coupled element $e = (r_e, c_e)$ define the variable adjacency lists:

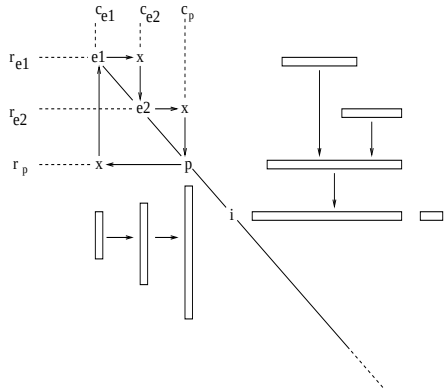
$$\begin{aligned} \mathcal{L}_e &\equiv \{r_i : (r_i, c_e) \in \bar{\mathcal{E}}\} \subseteq \mathcal{V}_r, \text{ the set of row variables adjacent to } c_e\,, \\ \mathcal{U}_e &\equiv \{c_j : (r_e, c_j) \in \bar{\mathcal{E}}\} \subseteq \mathcal{V}_c, \text{ the set of column variables adjacent to } r_e\,. \end{aligned}$$

In other words, $\mathcal{L}_e$ and $\mathcal{U}_e$ are respectively the sets of row and column vertices in the bi-clique induced after elimination of the coupled element e.

Now, suppose a pivot $p = (r_p, c_p)$ is chosen to be eliminated next. If there exists a cycle of the form $r_p \to c_{e_1} \to r_{e_1} \ldots \to c_{e_l} \to r_{e_l} \to c_p \to r_p$ (referred to as a *strongly connected component* in [20]), then, $\mathcal{L}_{e_i} \subseteq \mathcal{L}_p$ and $\mathcal{U}_{e_i} \subseteq \mathcal{U}_p$ for all i. Hence, except for (r_p, c_p), the other coupled elements in the cycle are not needed any more. See Figure 1(a) for an illustration. When updating the quotient graph, we can coalesce the coupled elements in the cycle into a single "supervertex", using the last element p as the representative vertex, and removing the other elements and the incident edges. This processed will be referred to as *element absorption*.

The transformation from bipartite quotient graph $\mathcal{G}^{k-1}$ to $\mathcal{G}^k$ at step k is carried out as follows. We search in the subgraph of $\mathcal{G}^{k-1}$ induced by $\bar{\mathcal{V}}_r^{k-1} \cup \bar{\mathcal{V}}_c^{k-1}$ for cycles that include the pivot (r_p, c_p). We then perform the element absorptions and form the new adjacency lists $\mathcal{L}_p$

(a) Quotient graph with a cycle.



(b) Quotient graph without cycle.

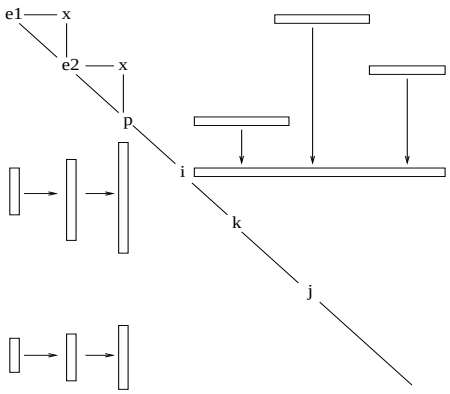


Figure 1: (a) Quotient graph with a cycle: $r_p \rightarrow c_{e_1} \rightarrow r_{e_1} \rightarrow c_{e_2} \rightarrow r_{e_2} \rightarrow c_p$. (b) Quotient graph without cycle.

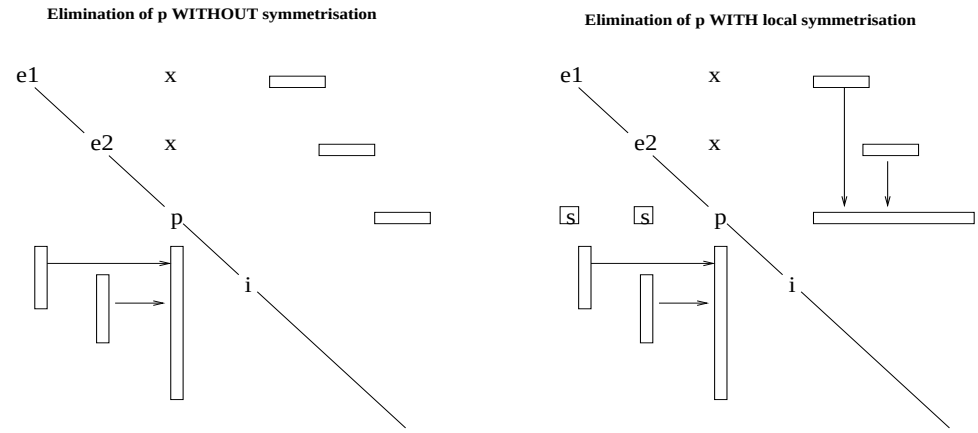
and $\mathcal{U}_p$. The structure of a column k in the reduced submatrix, $\mathbf{L}_{*k}$, can be determined very easily using $\mathcal{G}^{k-1}$: $\ell_{ik} \neq 0$ if and only if r_i is reachable from c_k through the coupled elements in $\mathcal{G}^{k-1}$. The structure of $\mathbf{U}_{k*}$ can be determined in a similar way. The bi-clique introduced by the current pivot is then used to prune the edges in $\mathcal{E}^k$. This process will be referred to as *variable pruning*. From the variable pruning process it results that $(r_i, c_j) \in \mathcal{E}^k$ if and only if $(r_i, c_j) \in E$ and entry $a_{i,j}$ of the original matrix has not been modified during steps 1 through k of the elimination. It was proved that using this scheme, the in-place property is maintained for each $\mathcal{G}^k$ [20]. But unlike the symmetric case, here, computing the reachable sets can be very expensive, because the length of the search path is not bounded by two. (In fact, it can be as long as $|\bar{\mathcal{V}}_r \cup \bar{\mathcal{V}}_c| + 1$ if no cycle is found.) This is illustrated by the example in Figure 1(b). There is only a simple path between p and elements e_1 and e_2, which results in $\mathcal{L}_{e_1} \subseteq \mathcal{L}_{e_2} \subseteq \mathcal{L}_p$. However, $\mathcal{U}_{e_1} \not\subseteq \mathcal{U}_p$ and $\mathcal{U}_{e_2} \not\subseteq \mathcal{U}_p$. For an uneliminated variable i such that $r_i \in \mathcal{L}_p \cap \mathcal{L}_{e_1} \cap \mathcal{L}_{e_2}$, all the column variables in $\mathcal{U}_p \cup \mathcal{U}_{e_2} \cup \mathcal{U}_{e_1}$ need to be included in $\mathbf{U}_{i*}$. In an in-place algorithm, one must store r_i only in $\mathcal{L}_{e_1}$, and then via the path $c_{e_1} \rightarrow r_{e_1} \rightarrow c_{e_2} \rightarrow r_{e_2} \rightarrow c_p$ one can deduct that $\mathbf{U}_{i*}$ should contain the union $\mathcal{U}_p \cup \mathcal{U}_{e_2} \cup \mathcal{U}_{e_1}$. We also note that, although $\mathcal{L}_{e_1} \subseteq \mathcal{L}_{e_2} \subseteq \mathcal{L}_p$, $\mathcal{L}_{e_1}$ alone may be required to build $\mathbf{L}_{*j}$ for any j such that $c_j \in \mathcal{U}_{e_1}$ and $c_j \notin \mathcal{U}_{e_2} \cup \mathcal{U}_p$. This means that $\mathcal{L}_{e_1}$ should not be absorbed into $\mathcal{L}_p$. Furthermore, if one considers a variable k such that $c_k \in \mathcal{U}_p$ but $c_k \notin \mathcal{U}_{e_1} \cup \mathcal{U}_{e_2}$, then $\mathcal{L}_p$ will need be included in $\mathcal{L}_k$. If we maintain the in-place property, the entries belonging to both $\mathcal{L}_{e_i}$, $i = 1, 2$, and $\mathcal{L}_p$ are only stored in $\mathcal{L}_{e_i}$, then we must be able to reach e_i, $i = 1, 2$ through a path starting at p: $c_p \rightarrow r_{e_2} \rightarrow c_{e_2} \rightarrow r_{e_1}$.

2.3 Local symmetrization

To avoid the long search path in a truly nonsymmetric algorithm, we have designed a relaxed diagonal Markowitz scheme. Figure 2 illustrates such a relaxation. The entry marked $\boxed{\text{s}}$ shows an artificial nonzero introduced to symmetrize only a *local* part of the matrix. In the example, we assume that (r_p, c_p) is the current pivot, and $\mathcal{R}_p = \emptyset$ and $\mathcal{C}_p = \{r_{e_1}, r_{e_2}\}$. We also assume that $\mathcal{U}_{e_1} \not\subseteq \mathcal{U}_p$ and $\mathcal{U}_{e_2} \not\subseteq \mathcal{U}_p$. (For the sake of clearness, we have even assumed that $\mathcal{U}_{e_1} \cap \mathcal{U}_p = \emptyset$ and $\mathcal{U}_{e_2} \cap \mathcal{U}_p = \emptyset$.) In order to obtain the row structure $\mathbf{U}_{i*}$, where $r_i \in \mathcal{L}_{e_1} \cap \mathcal{L}_{e_1} \cap \mathcal{L}_p$, $\mathcal{R}_i$ must contain c_{e_1}, c_{e_2}, and c_p. In other words, all the variables in $\mathcal{U}_{e_1} \cup \mathcal{U}_{e_2} \cup \mathcal{U}_p$ should be included

in $\mathbf{U}_{i*}$. With symmetrization (shown on the right of Figure 2), we pretend that $\mathcal{R}_p = \{c_{e_1}, c_{e_2}\}$ and $\mathcal{C}_p = \{r_{e_1}, r_{e_2}\}$. Therefore, $\mathcal{U}_{e_1} \subseteq \mathcal{U}_p$ and $\mathcal{U}_{e_2} \subseteq \mathcal{U}_p$. Hence, the coupled element p can absorb the coupled elements e_1 and e_2. As a result, we now only need the adjacency lists of r_p and c_p to get the adjacency lists of r_i and c_i. This eliminates the need to keep the adjacency lists of r_{e_1}, r_{e_2}, c_{e_1}, and c_{e_2}.

Figure 2: Illustration of local symmetrization.



In summary, the local symmetrization works as follows. Suppose the current pivot is (r_p, c_p). The adjacency lists $\mathcal{U}_p$ and $\mathcal{L}_p$ are computed by

$$\mathcal{U}_p = \left(\mathcal{A}_{p*} \cup \bigcup_{e \in \mathcal{R}_p} \mathcal{U}_e \cup \bigcup_{e \in \mathcal{C}_p} \mathcal{U}_e \right) \backslash \{c_p\} \,, \tag{3}$$

$$\mathcal{L}_p = \left(\mathcal{A}_{*p} \cup \bigcup_{e \in \mathcal{C}_p} \mathcal{L}_e \cup \bigcup_{e \in \mathcal{R}_p} \mathcal{L}_e \right) \backslash \{r_p\} \,. \tag{4}$$

The third terms in the unions results from the local symmetrization. The adjacency lists in the bipartite quotient graph (see Equations 1 and 2) of all the row (column) variables in the adjacency lists of the newly formed coupled element p should then be updated. All the row and column elements in $\mathcal{R}_p \cup \mathcal{C}_p$ are absorbed by the coupled element p. Therefore, if (r_e, c_e) is such an absorbed element, then r_e (c_e) will be replaced by r_p (c_p) each time it appears in an edge of $\bar{\mathcal{E}}$ and will be excluded from the quotient graph together with $\mathcal{L}_e$ $(\mathcal{U}_e)$. Furthermore, because of local symmetrization, more variable pruning can be performed. Let $i = (r_i, c_i)$ be a coupled variable (diagonal entry in the reduced matrix) such that $r_i \in \mathcal{L}_p$ and $c_i \notin \mathcal{U}_p$. We can anticipate local symmetrization between i and the coupled element p to prune all the row variables in $\mathcal{A}_{*i}$ that belong to $\mathcal{L}_p$. Entries in $\mathcal{A}_{i*}$ can also be pruned in a similar way (even if $r_i \notin \mathcal{L}_p$).

Our relaxation mechanism will be referred to as *local symmetrization* because the symmetrization is applied to only the local part of the graph involving only those row and column elements adjacent to c_p and r_p. Globally, the nonzero structure generally still remains nonsymmetric: (the index sets $\{k : r_k \in \mathbf{L}_{*i}\}$ and $\{k : c_k \in \mathbf{U}_{i*}\}$ are different). By construction, the length of a search path is bounded by three. In essence, we trade off some amount of asymmetry and space (because some zero entries may be stored) with a much faster search algorithm. We

show in Theorem 2.1 that although the local symmetrization may introduce extra (zero) entries in the factors with respect to a pure nonsymmetric scheme (see Figure 2), it leads to an in-place algorithm.

Theorem 2.1 *Let v denote a row or a column variable in $\mathcal{G}^k$ then for all $2 \leq k \leq n$ $|\mathcal{A}^k_{v*}| + |\mathcal{R}^k_v| \leq |\mathcal{A}^{k-1}_{v*}| + |\mathcal{R}^{k-1}_v| \leq |\mathcal{A}^0_{v*}|$, and $|\mathcal{A}^k_{*v}| + |\mathcal{C}^k_v| \leq |\mathcal{A}^{k-1}_{*v}| + |\mathcal{C}^{k-1}_v| \leq |\mathcal{A}^0_{*v}|$*

Proof. We focus on the row structures in this proof. The proof for the column structures is similar. We prove this theorem by induction. By construction, $\mathcal{R}^0_i = \emptyset$, so $|\mathcal{A}^0_{i*}| + |\mathcal{R}^0_i| = |\mathcal{A}^0_{i*}|$. Suppose at the kth step of elimination, (r_p, c_p) is selected as pivot. We first build $\mathcal{U}^k_p$ using Equation (3). The entries in $\mathcal{U}^k_p$ either come from the original matrix $\mathcal{A}_{p*}$ or from the entries in $\mathcal{U}_e$ such that $e \in \mathcal{R}_p \cup \mathcal{C}_p$. Because of local symmetrization, the coupled element e will be absorbed by p and the space of $\mathcal{U}_e$ can be used by $\mathcal{U}^k_p$ to store the new entries from $\mathcal{U}_e$. To take into account the fill-in we have to update the adjacency lists of all variables adjacent to the pivot. We focus on the row structures and thus consider the updating of $\mathcal{U}^k_i$ for $r_i \in \mathcal{L}^k_p$ using Equation (1). By construction, all the entries in $\mathcal{U}^k_p \cap \mathcal{A}^{k-1}_{i*}$ are pruned from $\mathcal{A}^{k-1}_{i*}$, showing that $|\mathcal{A}^k_{i*}| \leq |\mathcal{A}^{k-1}_{i*}|$, and c_p is added to $\mathcal{R}^k_i$. Now we consider the size of $\mathcal{R}^k_i$. There are two cases: 1) If c_p was in $\mathcal{A}^{k-1}_{i*}$, then it is removed from $\mathcal{A}^{k-1}_{i*}$ and added to $\mathcal{R}^k_i$, resulting in $|\mathcal{A}^k_{i*}| + |\mathcal{R}^k_i| \leq |\mathcal{A}^{k-1}_{i*}| + |\mathcal{R}^{k-1}_i|$; 2) If c_p was not in $\mathcal{A}^{k-1}_{i*}$, then the entry (r_i, c_p) must have been modified when eliminating an earlier pivot. Therefore there exists a coupled element e unabsorbed at step $k-1$ such that $r_i \in \mathcal{L}_e$ and $c_p \in \mathcal{U}_e$. We thus have $r_i \notin \mathcal{A}^{k-1}_{*p}$ and $e \in \mathcal{R}^{k-1}_i$ and $e \in \mathcal{C}^{k-1}_p$. Because of local symmetrization, e is absorbed by p at step k. Entry e in $\mathcal{R}^{k-1}_i$ can then be replaced by p, resulting in $|\mathcal{R}^k_i| \leq |\mathcal{R}^{k-1}_i|$. Thus, for all the cases, we have $|\mathcal{A}^k_{i*}| + |\mathcal{R}^k_i| \leq |\mathcal{A}^{k-1}_{i*}| + |\mathcal{R}^{k-1}_i|$. □

Corollary 2.2 *The quotient graphs $\mathcal{G}^k$ generated at each step of the diagonal Markowitz scheme with local symmetrization can be stored in the space of the original graph $\mathcal{G}^0$. More precisely, $|\mathcal{E}^k \cup \bar{\mathcal{E}}^k| \leq |\mathcal{E}^{k-1} \cup \bar{\mathcal{E}}^{k-1}| \leq |\mathcal{E}^0|$, for all $2 \leq k \leq n$.*

Theorem 2.1 implies that the in-place property holds for the row adjacency lists and for the column adjacency lists so that we could have an in-place implementation while keeping two separate lists to store entries in rows and in columns at each step of the elimination.

We will call our relaxed scheme *Diagonal Markowitz with Local Symmetrization* (`DMLS`). We now illustrate its main properties with an example. In Figure 3, we apply the `DMLS` algorithm assuming that pivots are in the natural ordering. The matrix on the right is the structure of the **LU** factors. The elimination tree built by the `DMLS` algorithm is shown in Figure 4. Each node of the tree corresponds to the elimination of a pivot. The nonsymmetric frontal matrix of each node corresponds to the structure of $\mathcal{U}_p$ and $\mathcal{L}_p$ as defined by Equations (3) and (4). The dark area corresponds to the entries in the reduced matrix updated during the node elimination (i.e. the nonsymmetric contribution block sent by one node to its parent).

At the first step, pivot $(1,1)$ is eliminated resulting in two fill-ins (**F** in positions $(4,2)$ and $(4,3)$). In the quotient graph $\mathcal{G}^1$, these fill-ins are implicitly represented by removing r_1 from $\mathcal{A}_{*2}$ and $\mathcal{A}_{*3}$, and adding r_1 to $\mathcal{C}_2$ and $\mathcal{C}_3$. Note that at this step there is no symmetrization of the column and row adjacency lists of r_1 and c_1, which otherwise would result in a completely full reduced matrix. When eliminating pivot $(2,2)$, since $r_1 \in \mathcal{C}_2$ and $c_1 \notin \mathcal{R}_2$, local symmetrization is applied and when computing $\mathcal{U}_2$ by Equation (3) entry **S** in position $(2,3)$ is added to the quotient graph $\mathcal{G}^2$ (i.e., the coupled element 1 is absorbed by 2, and r_2 is added to $\mathcal{C}_3$). One should note that entry $(2,1)$ is only *virtually* considered as non-zero and is never added in the

Figure 3: Illustration of the fill-ins introduced by the DMLS algorithm. **F** corresponds to the normal fill-in when eliminating pivot $(1,1)$. **S** corresponds to the fill-in due to the local symmetrization when eliminating pivot $(2,2)$.

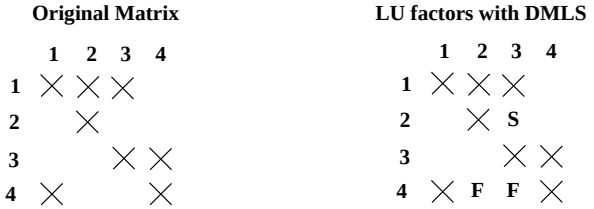
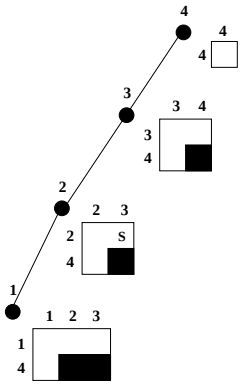


Figure 4: Elimination tree built by the DMLS algorithm applied to the matrix of Figure 3



LU factors. Similarly, when eliminating pivot $(3,3)$ at the next step, only the effect of adding entry $(3,2)$, by symmetrization of $(2,3)$, on the structure of the column $\mathcal{L}_3$ is considered (it happens to have no effect in our example). Even if entry $(3,2)$ is not effectively stored and has no effect on the size of the factors, it still has an effect on the structure of the dependency graph, as shown in Figure 4. The fact that pivot 3 can absorb the coupled element 2 because of the artificial $(3,2)$ nonzero entry also means that node 3 in Figure 4 becomes the unique parent of node 2 in the dependency graph, which in turn becomes a tree (or forest when the matrix is reducible). It is also interesting to note that, since entry $(2,3)$ (**S** in the figure) is considered nonzero, column 3 is added to the frontal matrix of node 2. But entry $(4,3)$ will not be modified during elimination of pivot $(2,2)$ because entry $(2,3)$ is structurally zero. Entry $(4,3)$ is a contribution resulting only from elimination of pivot $(1,1)$, and it is needed only when eliminating pivot $(3,3)$. Because of this newly added column, the frontal matrix of node 2 has the minimum structure to carry all the contributions of node 1 to all of its ancestral nodes 2 and 3. The edge between nodes 1 and 3 can be removed which corresponds to the coupled element 2 absorbing the coupled element 1 in the quotient graph.

We have shown that even if local symmetrization may result in extra fill-ins, it does not symmetrize the adjacency lists of the pivot; it builds at each elimination step the minimal nonsymmetric structure capable of absorbing all the nonsymmetric contributions from all the elements adjacent to the pivot. This nonsymmetric structure is called the *nonsymmetric frontal matrix*, similar to the symmetric case. By doing so, node p becomes the *unique* parent of all the nodes e such that $c_e \in \mathcal{R}_p$ or $r_e \in \mathcal{C}_p$ in a tree rooted with the last pivot. The DMLS algorithm

thus explicitly builds an elimination tree in which each node corresponds to the processing of a nonsymmetric frontal matrix whose structure is defined by $\mathcal{L}_p$ and $\mathcal{U}_p$. This elimination tree is identical to the dependency graph that MA41_UNS [6] would build if the same ordering were provided. In fact, the DMLS ordering is searching for an ordering that provides a good nonsymmetric elimination tree with respect to some local criterion/metric. The DMLS ordering also provides a good estimation of the size of the factors and all the working space required during numerical factorization using the MA41_UNS approach. This estimation is exact if the diagonal pivots are numerically stable.

2.4 The DMLS algorithm

To design the DMLS algorithm, we have exploited many algorithmic techniques from the AMD approach [1], and have extended them to the nonsymmetric case. The main difficulty is handling local symmetrization during degree calculation. We first explain how to adapt the symmetric algorithms, then describe the modifications needed for local symmetrization, and conclude this section with a description of the metrics used in pivot selection.

Exploiting identical structures in the graph can greatly speed up the degree update at each elimination step. Two coupled variables $i = (r_i, c_i)$ and $j = (r_j, c_j)$ are said to be *indistinguishable* in $\mathcal{G}$ if they have the same row adjacency structure **and** the same column adjacency structure in $\mathcal{G}$. (Although the row structure may be different from the column structure.) Indistinguishable coupled variables can then be merged into a single so-called *supervariable*. We use boldface letter to denote a supervariable. Thus, $\mathbf{i} = (\mathbf{r}_i, \mathbf{c}_i)$, with $\mathbf{r}_i \equiv \{r_i, r_j\}$, and $\mathbf{c}_i \equiv \{c_i, c_j\}$.

For each row supervariable $\mathbf{r}_i$, let d_{r_i} denote its external row degree [1, 17]. Similarly, for each column supervariable $\mathbf{c}_i$, let d_{c_i} denote its external column degree. The external degrees are defined as

$$d_{r_i} = |\mathcal{A}_{i*} \backslash \mathbf{c}_i| + |\,(\bigcup_{e \in \mathcal{R}_i \cup \mathcal{C}_i} \mathcal{U}_e) \setminus \mathbf{c}_i\,|\,, \tag{5}$$

$$d_{c_i} = |\mathcal{A}_{*i} \backslash \mathbf{r}_i| + |\,(\bigcup_{e \in \mathcal{C}_i \cup \mathcal{R}_i} \mathcal{L}_e) \setminus \mathbf{r}_i\,|\,. \tag{6}$$

Note that we should consider **all** the elements in **both** $\mathcal{R}_i$ and $\mathcal{C}_i$ contributing to the row degree and column degree. Indeed, because of local symmetrization, when (r_i, c_i) is selected as pivot at a later step, those elements will contribute to the structure of both $\mathcal{U}_i$ and $\mathcal{L}_i$ (see Equations (3) and (4)). Therefore, we must ensure that the computed degrees are also consistent with the local symmetrization scheme. This does not mean that we symmetrize all the edges in $\bar{\mathcal{E}}$. It only means that our degree evaluation must anticipate what would happen if (r_i, c_i) were selected as pivot. That is, during the degree calculation of the uneliminated variables, we need to simulate the effect of local symmetrization. The local symmetrization actually takes place only when a variable is selected as pivot. This has been illustrated in Figure 3.

Following the symmetric AMD algorithm [1], we can approximate the true degrees by their upper bounds, $\bar{d}_{r_i}$ and $\bar{d}_{c_i}$, which, at step k, can be computed by

$$\bar{d_{r_i}}^{\,k} = \min \begin{cases} n - k \\ \bar{d_{r_i}}^{\,k-1} + |\mathcal{U}_p \backslash \mathbf{c}_i| \\ |\mathcal{A}_{i*} \backslash \mathbf{c}_i| + |\mathcal{U}_p \backslash \mathbf{c}_i| + \sum_{e \in \mathcal{R}_i \cup \mathcal{C}_i} |\mathcal{U}_e \backslash \mathcal{U}_p| - \alpha_i \end{cases} \tag{7}$$

$$\bar{d_{c_i}}^{k} = \min \begin{cases} n-k \\ \bar{d_{c_i}}^{k-1} + |\mathcal{L}_p \backslash \mathbf{r}_i| \\ |\mathcal{A}_{*i} \backslash \mathbf{r}_i| + |\mathcal{L}_p \backslash \mathbf{r}_i| + \sum_{e \in \mathcal{C}_i \cup \mathcal{R}_i} |\mathcal{L}_e \backslash \mathcal{L}_p| - \beta_i \end{cases} \tag{8}$$

Note that, unlike the symmetric case, two correction terms α_i and β_i have been introduced to improve the accuracy of the approximation to the external degree. Let us justify the α_i term in Equation (7). In the unsymmetric case, it may happen that $\mathbf{c}_i \notin \mathcal{U}_p$, whereas for an accurate prediction of $\bar{d_{r_i}}$ in the context of local symmetrization, we need to pretend that $\mathbf{c}_i \in \mathcal{U}_p$. In this case, $|\mathbf{c}_i|$ was mistakenly counted in every $|\mathcal{U}_e \backslash \mathcal{U}_p|$ for $e \in \mathcal{C}_i$, and should thus be deducted. The total amount that should be deducted is $\alpha_i = |\mathcal{C}_i| \times |\mathbf{c}_i|$, see [4] for details.

Once $\bar{d_{r_i}}$ and $\bar{d_{c_i}}$ are computed, we have many choices of minimization criteria to select the next pivot. Each choice will lead to a different ordering. One set of criteria or metrics is degree-based, which is a direct function of the degrees (e.g., $Min(\bar{d_{r_i}} \times \bar{d_{c_i}})$, $Min(\bar{d_{r_i}} + \bar{d_{c_i}})$, $Min(Min(\bar{d_{r_i}}, \bar{d_{c_i}}))$ $Min(Max(\bar{d_{r_i}}, \bar{d_{c_i}}))$). Another set is deficiency-based, which is based on estimates of the amount of new fill-in generated at each step. We have experimented several variants of the approximations of the deficiency. Most of the heuristics in [19, 22] can be adapted easily to the unsymmetric case. Moreover, we have considered a deficiency heuristic that results from discussions with T. Davis and I.S Duff while working on the approximate minimum degree ordering for symmetric matrices `AMD`. This approximation of the deficiency (referred to as `AMDF` in the symmetric context) is based on the following observation. Suppose $\{r_p, c_p\}$ is the current pivot and the two column elements e_1 and e_2 are adjacent to $r_i \in \mathcal{L}_p$. In our approximate degree $\bar{d_{c_i}}$ we count twice the row variables that belong to $(\mathcal{L}_{e_1} \backslash \mathcal{L}_p) \cap (\mathcal{L}_{e_2} \backslash \mathcal{L}_p)$. This property can be exploited to improve the estimation of the deficiency since, in this context, we try to deduct from the degree product the cliques of all the elements adjacent to the current variable. We can consider that $(\mathcal{L}_{e_1} \backslash \mathcal{L}_p) \cap (\mathcal{L}_{e_2} \backslash \mathcal{L}_p) = \emptyset$, because this overlapped term also occurs in the degree product, which is cancelled after subtraction. Thus, for each $r_i \in \mathcal{L}_p$, we can deduct both the area relative to the current clique p (i.e. $|L_p| \times |U_p|$) and the sum of the "external areas" of all the elements adjacent to (r_i, c_i) (i.e. $\sum_{e \in \mathcal{C}_i \cup \mathcal{R}_i} |\mathcal{L}_e \backslash \mathcal{L}_p| \times |\mathcal{U}_p|$) The external area is readily available since $|\mathcal{L}_e \backslash \mathcal{L}_p|$ has already been computed during the approximate degree calculation. This leads to a more accurate approximation of the deficiency than the approximations introduced in [19, 22] when used in an approximate minimum degree code. This approximation of the deficiency can be easily adapted to our nonsymmetric ordering and will be referred to as `DMLS-MF`. Note that using `AMDF` on symmetric matrices, the amounts of reduction in fill-in and flop count relative to `AMD` have been found similar to those reported in [19, 22].

3 Numerical experiments

We now evaluate the `DMLS` ordering algorithm and compare its ordering quality with that obtained by applying both minimum degree and minimum deficiency algorithms on $\mathbf{A} + \mathbf{A}^T$.

3.1 Testing environment

To experiment with our ordering algorithm, we will consider the unsymmetrized multifrontal code `MA41_UNS` [2, 6], which automatically detects and exploits the structural asymmetry of the submatrices involved when processing the elimination tree associated with the pattern of the symmetric matrix $\mathbf{A} + \mathbf{A}^{\mathrm{T}}$. In [7], `MA41_UNS` with `AMD` ordering was shown to be very competitive

with SuperLU and UNFPACK on a large class of matrices including very unsymmetric ones. We will show in this section that using DMLS ordering can significantly improve the speed of MA41_UNS. MA41_UNS is a tree-based multifrontal algorithm, in which some steps of Gaussian elimination are performed on a dense frontal matrix at each node of the assembly tree and the Schur complement (or the contribution block) that remains is passed for assembly at the parent node.

MA41_UNS can benefit from a numerical preordering (row or column permutations) to maximize the magnitude of the diagonal entries and from a numerical scaling of the matrix. After numerical pivoting and scaling, a sparsity preserving ordering (symmetric permutation of $\mathbf{A}$) based on an analysis of the pattern of $\mathbf{A} + \mathbf{A}^T$ can be used. The computational graph of the factorization is then computed assuming that diagonal pivots are numerically stable. Since this assumption may not be entirely true during numerical factorization, the solver has the mechanisms to accommodate for element growth to some extent. Standard partial pivoting with a threshold value is used to select numerically stable pivots. It is thus possible that some variables cannot be eliminated from a frontal matrix. The rows and columns containing the non-eliminated variables of a frontal matrix are then added to the contribution block and passed to the parent node. Those *delayed* eliminations will result in an increase in the size of the $\mathbf{LU}$ factors estimated in the analysis and an increase in the number of operations. In practice, it has been observed that using MC64 [9, 10] from HSL [15] as preordering can significantly reduce the number of delayed pivots during factorization [3]. This preordering will thus be applied on all our test matrices.

Our test matrices are from the forthcoming Rutherford-Boeing Sparse Matrix Collection [8], the industrial partners of the PARASOL Project[1], Tim Davis' collection[2], and SPARSEKIT2[3]. Only matrices with structural symmetry less than 0.5 and dimension greater 1000 were chosen. We define the structural symmetry as the fraction of the nonzeros matched by nonzeros in symmetric locations. Thus, a symmetric matrix has a value of 1, and a highly nonsymmetric matrix has a value close to 0. When there were many similar matrices from the same application domain, we only used a subset with largest dimensions. Altogether, there were 61 structurally nonsymmetric matrices in our study.

Our computer platform comprises of a 2.8 GHz Pentium 4 processor, 2 GBytes of memory and 1 MByte of cache, with Linux operating system. We used `gcc -O` to compile the DMLS code and `pgf90 -O` to compile all the FORTRAN routines. We also used Goto's BLAS library `libgoto_p4_512-r0.94.so` [14].

We systematically applied random row and column permutations to each matrix. Eleven different permutations were applied to each matrix and the run that provided the median value of the $\mathbf{LU}$ factor size was used in the report.

3.2 Results

We first evaluated the quality of the DMLS ordering when using different minimization metrics and heuristics mentioned in Section 2.4 (min-prod, min-sum, min-min, min-max, and minimum deficiency). Our study showed that DMLS-MF (i.e., DMLS with approximate minimum deficiency) gives the best quality in terms of fill-in and flop reductions. Therefore, we used DMLS-MF in the rest of the experiments. To illustrate the gain in quality we compared DMLS-MF with the

[1] EU ESPRIT IV LTR Project 20160

[2] Web page `http://www.cise.ufl.edu/research/sparse/matrices`

[3] Web page `http://math.nist.gov/MatrixMarket/data/SPARSKIT/`

standard approximate minimum degree algorithm AMD as well as AMDF (our best local heuristic to approximate the deficiency for the symmetrized matrix $\mathbf{A} + \mathbf{A}^T$).

We observed that for five highly reducible matrices (raefsky5.rua, raefsky6.rua, meg1.rua, bayer05.rua and bayer07.rua) DMLS-MF significanly outperformed both AMD and AMDF—the factor sizes were reduced by 4 to 10 times. Although this is a nice property of DMLS, we have excluded these five matrices when reporting the results, because they will skew the statistics. For the other 56 matrices, we compare in Figure 5 the actual size of the factors (including the extra fill-ins due to numerical pivoting) of the DMLS-MF, AMD and AMDF orderings. For a relatively large number of matrices (28 with respect to AMD and 23 with respect to AMDF), the DMLS-MF ordering leads to more than 20% reduction in the size of the factor. The best reduction is more than a factor of 2. Sometimes DMLS-MF may give worse ordering than AMD or AMDF, but it is no more than 30% worse. Note that there are 8 matrices which have structural symmetry less than 0.5 initially but larger than 0.5 after preordering with MC64. As expected, for these matrices, relatively smaller gains are obtained from DMLS.

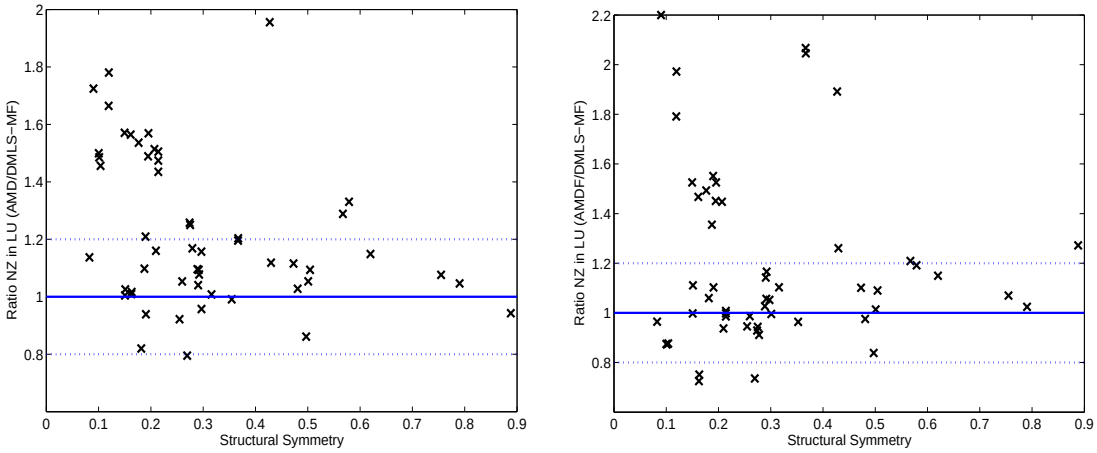


Figure 5: Actual fill-in ratios. The X-axis shows the structural symmetry after preprocessing. Left figure: AMD/DMLS-MF, mean gain is 22%, median value is 14%; Right figure: AMDF/DMLS-MF, mean gain is 20%, median value is 6%.

In Figure 6, we compare the number of floating-point operations performed during factorization (including numerical pivoting) using the three orderings. For a large number of matrices, the DMLS-MF ordering leads to more than 30% flop reduction compared to AMD (39 matrices) and AMDF (28 matrices). Futhermore, for a significant number of matrices, the flop reduction with DMLS-MF is more than a factor of 2. The best reduction is almost a factor of 8 (matrix orani678.rua, not shown in the plots). Even when DMLS-MF is worse, it is no more than 2.5 times worse.

We now focus on 18 large matrices of dimension larger than 10000 and having initial structural symmetry smaller than 0.5 (except for Sandia/mult_dcop_03). This is a subet of the 61 matrices studied above. For this subset, we perform a more detailed quantitative comparison of the AMDF and DMLS-MF algorithms. These matrices are listed in Table 1, and are sorted in increasing symmetry after the matrices are randomly permuted and reordered using the maximum transversal given by MC64. Here, among the 11 symmetry numbers from the 11 initial random

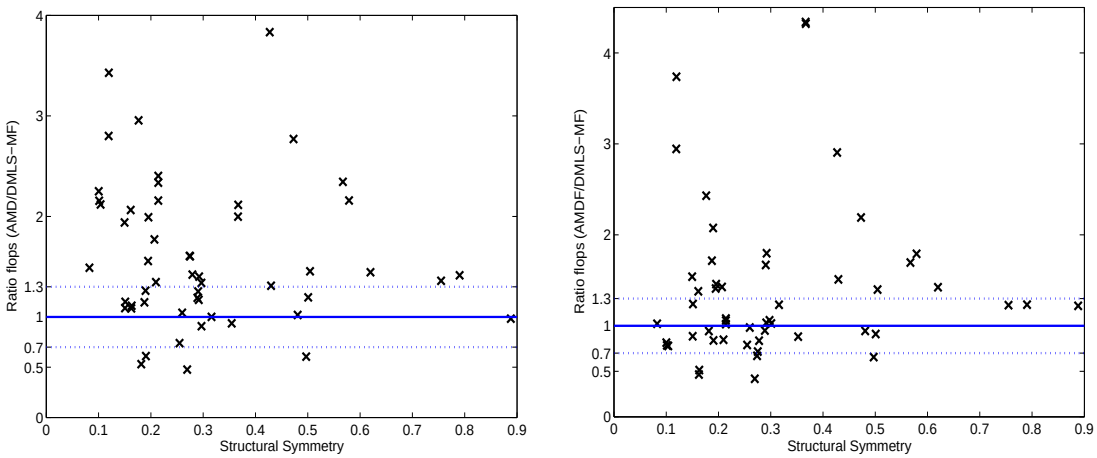


Figure 6: Ratio of the number of floating-point operations in factorization. Left figure: AMD/DMLS-MF, mean gain is 64%, median value is 39%; Right figure: AMDF/DMLS-MF, mean gain is 56%, median value is 17%; Note that matrix orani678 was excluded from the two plots because its flop reduction is almost 8 when compared to AMDF.

permutations, we report the one corresponding to the permutation that gives the mean fill ratio of AMDF over DMLS-MF.

In Table 2 we report both the estimated factor size given by the analysis phase (columns 2 and 3) and the actual factor size computed during factorization using MA41_UNS (columns 5 and 6). Since the pruned frontal matrix structures appeared in factorization are exactly those on which the DMLS algorithm is based, the estimation given by DMLS-MF is correct modulo small variation due to numerical pivoting. In fact, in addition to an ordering, DMLS also gives an assembly tree with the correct fronal size that MA41_UNS can use. It is important to note that numerical pivoting has little effect on the structural changes. But this is not the case with AMDF, which is based on the graph of the symmetrized matrix $\mathbf{A} + \mathbf{A}^T$. We see that the difference between estimation and actual size is significant, and the estimatation is often much larger than the actual size. This is because the MA41_UNS factorization algorithm can dynamically exploit a more precise frontal matrix structure at each pivot, which can be rectangular and smaller than the frontal matrix structure predicted by AMDF. (The frontal matrix predicted by AMDF is always square due to initial, global symmetrization $\mathbf{A} + \mathbf{A}^T$.) Furthermore, it has been observed in [6] that even larger difference can occur in the size of the stack memory. Therefore, after AMDF (or AMD) ordering and before numerical factorization, one should run an unsymmetric symbolic factorization algorithm to identify the unsymmetric structures needed to perform numerical factorization. In our context this extra cost should thus be added to the analysis time when an ordering based on $\mathbf{A} + \mathbf{A}^T$ is used.

In addition to the actual factor size and the floating-point operations, we also report the peak memory (labeled "Real memory" in Table 2) needed to factorize the matrix, which is measured in number of double precision words. For some classes of matrices (ATandT, Mallya, Norris, Sandia) the DMLS-MF ordering leads to much less memory usage than that of AMDF. For some other classes of matrices (Grund, Vavasis, Shen), the results are comparable. The results for Hollinger and Hohn matrices are comparable with the two orderings. We found that

Group/Matrix	n	nnz	StructSym		description
			Before	After	
Vavasis/av41092	41092	1683902	0.00	0.08	Unstructured finite element
Hollinger/g7jac200sc	59310	837936	0.10	0.10	Economic model
Hollinger/jan99jac120sc	41374	260202	0.00	0.16	Economic model
Mallya/lhr34c	35152	764014	0.00	0.19	Light hydrocarbon recovery
Mallya/lhr71c	70304	1528092	0.00	0.21	Light hydrocarbon recovery
Hollinger/mark3jac140sc	64089	399735	0.22	0.21	Economic model
Grund/bayer01	57735	277774	0.00	0.25	Chemical process simulation
Hohn/sinc18	16428	973826	0.01	0.27	Single-material crack problem (sinc-basis)
Hohn/sinc15	11532	568526	0.01	0.27	Single-material crack problem (sinc-basis)
Hohn/fd18	16428	63406	0.00	0.29	Finite difference approximation of crack problem
Sandia/mult_dcop_03	25187	193216	0.66	0.37	Circuit simulation
ATandT/twotone	120750	1224224	0.28	0.43	Harmonic balance method
ATandT/onetone1	36057	341088	0.10	0.43	Harmonic balance method
Norris/torso1	116158	8516500	0.43	0.43	Finite element matrices from bioengineering
Grund/poli_large	15575	33074	0.47	0.47	Chemical process simulation
Shen/shermanACb	18510	145149	0.26	0.50	Circuit simulation
ATandT/pre2	659033	5959282	0.36	0.58	Harmonic balance method
Shen/e40r0100	17281	553562	0.33	0.89	Fluid dynamics

Table 1: Test matrices. StructSym denotes the structural symmetry (both Before and After preprocessing).

the Hollinger matrices are very sensitive to the initial random permutations. For exemple, the number of operations varies between 6.5×10^8 and 8.3×10^8 using `AMDF`, between 12.7×10^8 and 18.4×10^8 using `DMLS-MF`, and between 17.1×10^8 and 20.7×10^8 using `AMD`. Moreover, for this class of matrices, `MA41_UNS` combined with the `AMD` ordering applied to $\mathbf{A} + \mathbf{A}^{\mathrm{T}}$ significantly outperforms all the unsymmetric solvers considered in [7]. Using `AMDF` thus further reduced the number of operations and the attempt to exploit the asymmetry of the original matrix did not improve the ordering quality (as shown by the `UMPFPACK` code which attempts to exploit all the asymmetry [7]).

For smaller matrices in the same classes, which are among the complete set of 61 matrices but not shown in Table 2, we have observed a similar behavior.

Finally, we report in Table 2 the runtimes of the ordering algorithms. Since both `AMDF` and `DMLS-MF` exploit approximate degree calculations, the complexity of these two codes is directly related to that of the `AMD` ordering. For `DMLS`, since we need to maintain the adjacency structures and the approximate degrees both row-wise and column-wise, we expect `DMLS-MF` to be twice slower than `AMDF`. This is in general true except for HOHN and NORRIS classes of matrices, for which `DMLS-MF` is much slower. For HOHN/SINC* matrices, large dense off-diagonal blocks lead to larger supervariables in the graph of $\mathbf{A}+\mathbf{A}^T$ than in the graph of $\mathbf{A}$. In this case, the asymmetry prevents `DMLS-MF` from selecting larger supervariables, whereas it is not sufficiently unsymmetric to lead to better ordering. For the matrix NORRIS/TORSO1, the situation is different for at least two reasons. Firstly, taking into account the asymmmetry of the matrix significantly improves the quality of the ordering. Secondly, it has been shown in our recent work [5] (generalization of the `DMLS` approach to allow off-diagonal and numerical-based pivot selection) that using separate row and column supervariables, one can significanlty decrease the ordering time on this class of matrices and this is true even when pivot selection is restricted to the diagonal as in `DMLS`.

However, considering separate row and column supervariables is not at all natural in the DMLS context; it would require significant modifications of the data structures used in DMLS code and is out of the scope of this work.

4 Summary

In this paper, we have considered the ordering problem for the triangular factorization of a sparse nonsymmetric matrix when pivots can be chosen on main diagonal. We have described a bipartite quotient graph model for nonsymmetric elimination, and have used it as a compact way to represent the elimination graph. The model was first proposed by Pagallo and Maulino [20], but to our knowledge, its implementation did not appear in any literature. Using this model, an ordering algorithm can be implemented in space bounded by the size of the original matrix. This is the so-called in-place property. However, we have found that a straightforward implementation may lead to an algorithm with much higher complexity than an AMD type of algorithm applied to the graph of $\mathbf{A} + \mathbf{A}^T$. In order to speed up the ordering algorithm itself, we have introduced the local symmetrization mechanism in the diagonal Markowitz scheme, which allows us to reduce the amount of backtracking needed to update the Schur complement structure at each step. As a result, we have obtained an efficient ordering algorithm both in space and in time—it has the in-place property and the same time complexity as the AMD type of algorithms.

We have performed numerical experiments on large numbers of matrices (61) that come from a wide range of applications. The results have showed that our modified diagonal Markowitz scheme indeed can produce better orderings. Compared to the best local greedy algorithms that cannot exploit asymmetry, our algorithm has achieved average reductions of 22% in factor size and 56% in flop count.

Matrix	Size of factors (10^6)						Flops (10^9)			Real memory (10^6)			Ordering time (seconds)		
	Estimated			Actual											
	AMF	DMLS	Ratio	AMF	DMLS	Ratio	AMF	DMLS	Ratio	AMF	DMLS	Ratio	AMF	DMLS	Ratio
av41092.rua	11.98	9.29	1.29	9.16	9.5	0.96	3.56	3.48	1.02	9.43	9.63	0.98	0.69	2.8	0.24
g7jac200sc.rua	30.62	29.84	1.03	26.12	29.87	0.87	25.25	30.98	0.82	26.94	30.88	0.87	4.75	9.31	0.51
jan99jac120sc.rua	4.5	4.29	1.05	3.11	4.29	0.73	0.77	1.66	0.46	3.14	4.53	0.69	1.23	2.81	0.44
lhr34c.rua	6.22	3.49	1.78	5.25	3.62	1.45	0.6	0.43	1.41	5.29	3.69	1.43	0.55	2.26	0.24
lhr71c.rua	12.77	7.21	1.77	10.75	7.43	1.45	1.27	0.89	1.43	10.77	7.49	1.44	1.24	5.31	0.23
mark3jac140sc.rua	19.34	14.89	1.3	14.9	14.93	1	7.68	7.26	1.06	15.26	15.41	0.99	1.38	4.23	0.33
bayer01.rua	2.51	1.99	1.26	1.88	1.99	0.95	0.09	0.11	0.79	1.88	2	0.94	0.59	1.22	0.49
sinc18.rua	36.76	31.65	1.16	29.28	31.72	0.92	40.84	60.22	0.68	30.11	35.39	0.85	0.98	10.84	0.09
sinc15.rua	18.03	15.39	1.17	14.37	15.47	0.93	14.31	21.35	0.67	14.79	17.54	0.84	0.49	4.33	0.11
Zhao2.rua	15.79	13.9	1.14	12.97	14.23	0.91	7.69	9.19	0.84	13.78	15.04	0.92	0.45	0.95	0.47
fd18.rua	1.44	1.05	1.38	1.1	1.07	1.03	0.11	0.12	0.95	1.11	1.12	0.99	0.1	0.18	0.53
mult_dcop_03.rua	2.73	0.94	2.9	1.87	0.91	2.07	0.51	0.12	4.34	1.96	0.97	2.02	3.98	0.43	9.26
twotone.rua	22.05	8.22	2.68	15.54	8.22	1.89	14.74	5.07	2.91	15.75	9.05	1.74	1.79	2.08	0.86
onetone1.rua	4.85	3.2	1.51	4.04	3.2	1.26	1.94	1.28	1.51	4.09	3.65	1.12	0.33	0.51	0.64
torso1.rua	41.29	34.08	1.21	41.28	34.19	1.21	58.69	35.91	1.63	42.34	36.62	1.16	1.7	69.42	0.02
poli_large.rua	0.06	0.03	1.7	0.04	0.03	1.1	0	0	2.19	0.04	0.03	1.11	0.02	0.01	2.16
shermanACb.rua	0.55	0.44	1.26	0.44	0.44	1.01	0.03	0.03	0.91	0.49	0.51	0.96	2.17	0.15	14.44
pre2.rua	115.58	90.12	1.28	107.53	90.23	1.19	405.93	226.6	1.79	161.32	99.98	1.61	17.03	61.2	0.28
e40r0100.rua	2.86	2.18	1.31	2.78	2.19	1.27	0.34	0.28	1.22	2.83	2.28	1.24	0.05	0.12	0.42

Table 2: Comparison of `DMLS-MF` and `AMDF` orderings.

References

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17:886–905, 1996.

[2] P. R. Amestoy and I. S. Duff. Vectorization of a multiprocessor multifrontal code. *Int. J. of Supercomputer Applics.*, 3:41–59, 1989.

[3] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Transactions on Mathematical Software*, 27(4):388–421, 2001.

[4] P. R. Amestoy, X. S. Li, and E. G. Ng. Diagonal markowitz scheme with local symmetrization. Technical Report LBNL-53854, Lawrence Berkeley National Laboratory, October 2003. Also ENSEEIHT-IRIT RT/APO/03/05.

[5] P. R. Amestoy, X. S. Li, and S. Pralet. Constrained Markowitz with local symmetrization. Technical Report TR/PA/04/137, CERFACS, Toulouse, France, 2004. Also appeared as ENSEEIHT-IRIT report RT/APO/04/05 and as Lawrence Berkeley Lab report LBNL-56861.

[6] P. R. Amestoy and C. Puglisi. An unsymmetrized multifrontal LU factorization. *SIAM J. Matrix Analysis and Applications*, 24:553–569, 2002.

[7] T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):165–195, 2004.

[8] I. S. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, 1997. Also Technical Report ISSTECH-97-017 from Boeing Information & Support Services and Report TR/PA/97/36 from CERFACS, Toulouse. `http://www.cse.clrc.ac.uk/Activity/SparseMatrices/`.

[9] I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Analysis and Applications*, 20(4):889–901, 1999.

[10] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Analysis and Applications*, 22(4):973–996, 2001.

[11] I.S. Duff, A.M. Erisman, and J. K Reid. *Direct Methods for Sparse Matrices*. Oxford University Press, Oxford, England, 1987.

[12] I.S Duff and J.K Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Mathematical Software*, 9(3):302–325, September 1983.

[13] Alan George and Joseph W. H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[14] Kazushige Goto. High-performance BLAS. http://www.cs.utexas.edu/users/flame/goto/.

[15] HSL. A collection of Fortran codes for large scale scientific computation. http://www.cse.clrc.ac.uk/Activity/HSL, 2000.

[16] Xiaoye S. Li and James W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.

[17] Joseph W.H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Mathematical Software*, 11:141–153, 1985.

[18] H.M. Markowitz. The elimination form of the inverse and its application to linear programming. *Management Sci.*, 3:255–269, 1957.

[19] E. Ng and P. Raghavan. Performance of greedy heuristics for sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 20:902–914, 1999.

[20] Giulia Pagallo and Consuelo Maulino. A bipartite quotient graph model for unsymmetric matrices. In *Lecture Notes in Mathematics 1005, Numerical Method*, pages 227–239, Springer-Verlag, New York, 1983.

[21] D. J. Rose and R. E. Tarjan. Algorithmics aspects of vertex elimination on directed graphs. *SIAM J. Appl. Math.*, 34:176–197, 1978.

[22] Edward Rothberg and Stanley C. Eisenstat. Node selection strategies for bottom-up sparse matrix ordering. *SIAM Journal on Matrix Analysis and Applications*, 19(3):682–695, 1998.